# UNIX - FILE PERMISSION / ACCESS MODES

 

File ownership is an important component of UNIX that provides a secure method for storing files. Every file in UNIX has the following attributes:

- **Owner permissions:** The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions:** The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions:** The permissions for others indicate what action all other users can perform on the file.

## The Permission Indicators:

While using **ls -l** command it displays various information related to file permission as follows:

```
$ls -l /home/amrood
-rwxr-xr--  1 amrood   users 1024  Nov 2 00:10  myfile
drwxr-xr--- 1 amrood   users 1024  Nov 2 00:10  mydir
```

Here first column represents different access mode ie. permission associated with a file or directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x):

- The first three characters (2-4) represent the permissions for the file's owner. For example -rwxr-x**r--** represents that onwer has read (r), write (w) and execute (x) permission.
- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example -rwxr-x**r--** represents that group has read (r) and execute (x) permission but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example -rwxr-x**r--** represents that other world has read (r) only permission.

## File Access Modes:

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which are described below:

## 1. Read:

Grants the capability to read ie. view the contents of the file.

## 2. Write:

Grants the capability to modify, or remove the content of the file.

## 3. Execute:

User with execute permissions can run a file as a program.

## Directory Access Modes:

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned:

## 1. Read:

Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.

## 2. Write:

Access means that the user can add or delete files to the contents of the directory.

## 3. Execute:

Executing a directory doesn't really make a lot of sense so think of this as a traverse permission.

A user must have execute access to the **bin** directory in order to execute ls or cd command.

## Changing Permissions:

To change file or directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod: symbolic mode and absolute mode.

## Using chmod in Symbolic Mode:

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

| Chmod operator | Description |
|---|---|
| **+** | Adds the designated permission(s) to a file or directory. |
| **-** | Removes the designated permission(s) from a file or directory. |
| **=** | Sets the designated permission(s). |

Here's an example using testfile. Running ls -1 on testfile shows that the file's permissions are as follows:

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example chmod command from the preceding table is run on testfile, followed by ls -l so you can see the permission changes:

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod u-x testfile
$ls -l testfile
-rw-rwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod g=r-x testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Here's how you could combine these commands on a single line:

```
$chmod o+wx,u-x,g=r-x testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

## Using chmod with Absolute Permissions:

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

| Number | Octal Permission Representation | Ref |
|---|---|---|
| **0** | No permission | --- |
| **1** | Execute permission | --x |
| **2** | Write permission | -w- |
| **3** | Execute and write permission: 1 (execute) + 2 (write) = 3 | -wx |
| **4** | Read permission | r-- |
| **5** | Read and execute permission: 4 (read) + 1 (execute) = 5 | r-x |
| **6** | Read and write permission: 4 (read) + 2 (write) = 6 | rw- |
| **7** | All permissions: 4 (read) + 2 (write) + 1 (execute) = 7 | rwx |

Here's an example using testfile. Running ls -1 on testfile shows that the file's permissions are as follows:

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example chmod command from the preceding table is run on testfile, followed by ls -l so you can see the permission changes:

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 043 testfile
$ls -l testfile
----r---wx  1 amrood   users 1024  Nov 2 00:10  testfile
```

## Changing Owners and Groups:

While creating an account on Unix, it assigns a owner ID and a group ID to each user. All the permissions mentioned above are also assigned based on Owner and Groups.

Two commands are available to change the owner and the group of files:

1. **chown:** The chown command stands for "change owner" and is used to change the owner of a file.
2. **chgrp:** The chgrp command stands for "change group" and is used to change the group of a file.

## Changing Ownership:

The chown command changes the ownership of a file. The basic syntax is as follows:

```
$ chown user filelist
```

The value of user can be either the name of a user on the system or the user id (uid) of a user on the system.

Following example:

```
$ chown amrood testfile
$
```

Changes the owner of the given file to the user **amrood**.

**NOTE:** The super user, root, has the unrestricted capability to change the ownership of a any file but normal users can change only the owner of files they own.

## Changing Group Ownership:

The chrgp command changes the group ownership of a file. The basic syntax is as follows:

```
$ chgrp group filelist
```

The value of group can be the name of a group on the system or the group ID (GID) of a group on the system.

Following example:

```
$ chgrp special testfile
$
```

Changes the group of the given file to **special** group.

## SUID and SGID File Permission:

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the **passwd** command, your new password is stored in the file /etc/shadow.

As a regular user, you do not have read or write access to this file for security reasons, but when you change your password, you need to have write permission to this file. This means that the **passwd** program has to give you additional permissions so that you can write to the file /etc/shadow.

Additional permissions are given to programs via a mechanism known as the Set User ID ( SUID) and Set Group ID ( SGID) bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is true for SGID as well. Normally programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter "s" if the permission is available. The SUID "s" bit will be located in the permission bits where the owners execute permission would normally reside. For example, the command

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x  1   root   bin  19031 Feb 7 13:47  /usr/bin/passwd*
$
```

Which shows that the SUID bit is set and that the command is owned by the root. A capital letter S in the execute position instead of a lowercase s indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users:

- The owner of the sticky directory
- The owner of the file being removed
- The super user, root

To set the SUID and SGID bits for any directory try the following:

```
$ chmod ug+s dirname
$ ls -l
drwsr-sr-x 2 root root  4096 Jun 19 06:45 dirname
$
```